Lecture 15
Perl Programming

Perl (Practical Extraction and Report Language) is a powerful and adaptable scripting language. Perl became very popular in early 90's as web became a reality. Perl is ideal for processing text files containing strings. Perl is also good for processing web pages containing tags of different types (image tags, url tags etc). These tags or substrings can be extracted using Perl commands. Perl programs are not compiled but interpreted. Perl interpreter in your unix system can be found by typing

- **where perl**

It may show
/usr/local/bin/perl
/usr/bin/perl

giving the path of the perl interpreter. Perl interpreter is used to run perl programs.

Let us start with a simple Hello world program in perl.

```
#!/usr/local/bin/perl
print "Hello World\n";
```

Assuming this is in a file called hello.pl, we can run the program by tying

- **perl hello.pl**

Or you can set the executable permission for the file and run the program as follows.

- **chmod +x hello.pl**
- **./hello.pl**

Strings in Perl

Perl strings can be surrounded by single quotes or double quotes. Single quote means string must be interpreted literally and double quotes could have "\n" type escape characters that have special meaning. So for example

print "hello world\n"; ➔ prints the string **hello world** with a new line
print 'hello world\n'; ➔ prints the string **hello world\n**

Scalars in Perl

A scalar in perl is either a number (103, 45.67) or a string. A string is a sequence of characters where each character is represented by 8-bits. There is also null string or the shortest possible string that has no characters. A string inside single quotes ('hello there') is a literal string, and double quoted strings can have escape characters such as '\t' (tab) inside them for formatting purposes. A double quoted string is very much like a C string.

Operators for Strings

Strings can be concatenated using "." Operator. So if we define two strings s1 and s2 and concatenate and store them in a string s3, you would do it like this in perl.

```
$s1 = "hello";
$s2 = "world";
$s3 = $s1.$s2;
```

Note that variable declarations are preceded by $. Other useful functions that can operate on strings are:

- **substr($s,n,m)** --- substring of $s from index n to m
- **index string, substring, position** – look for first index of the substring in string starting from position
- **index string, substring** – look for first index of the substring in string starting from the beginning
- **rindex string, substring** -- position of substring in string starting from the end of the string
- **length(string)** – returns the length of the string
- **$_ = string; tr/a/z/;** -- replaces all 'a' characters of string with a 'z' character. More variations available.
- **chop** – this removes the last character at the end of a scalar.
- **chomp** – removes a newline character from the end of a string
- **split –** splits a string and places in an array
    - **@array = split(/:/,$name);** # splits the string $name at each :
    - **The ASCII value of a character $a is given by ord($a)**

Comparison Operators

Comparison	Numeric	String
Equal	==	eq
Not Equal	!=	ne
Greater than	>	gt
Less than	<	lt
Greater or equal	>=	ge
Less or equal	<=	le

Another string operator of special interest is the letter x (lower case). This operator causes the variable to be repeated. For example,

```
$s1 = "guna";
$s2 = $s1 x 3;
```

will cause $s2 to store the string "gunagunaguna"

Operator Precedence and Associativity

Associativity	Operator
left	terms and list operators (leftward)
left	->
nonassoc	++ --
right	**
right	! ~ \ and unary + and -
left	=~ !~
left	* / % x
left	+ - .
left	<< >>
nonassoc	named unary operators (chomp)
nonassoc	< > <= >= lt gt le ge
nonassoc	== != <=> eq ne cmp
left	&
left	| ^
left	&&
left	||
nonassoc	.. ...
right	?:
right	= += -= *= etc.
left	, =>
nonassoc	list operators (rightward)
right	not
left	and
left	or xor

source: perl.com

Variables in Perl

We have already seen how to define a variable. Perl has three types of variables - scalars (strings or numeric's), arrays and hashes.
Let us look at defining scalar variables.

```
$x = 45.67;
$var = 'cost';
```

So a statement such as

```
print "$var is $x";
```

will print "cost is 45.67". Simple arithmetic can be performed on numeric variables such as

```
$x = 563;
$y = 32.56;
$y++;
$x += 3;
```

Arrays

Array in perl is defined as a list of scalars. So we can have arrays of numerics or strings. For example,

```
@array = (10,12,45);
@A = ('guna', 'me', 'cmu', 'pgh');
```

Defines arrays of numeric's and strings. To process the i^{th} element of an array A (array indices starts from 0) we simply refer to $A[i]. For example, we can write

```
$i = 1;
$A[$i] = 'guna';
```

this sets the element in A with index 1 to "guna".

The **length of an array A** can be found using $#A. The length of an array is one more than **$#A. That is**

```
$len = $#A + 1
```

You can also find length of an array as

```
$len = @A;
```

To **resize an array**, we can simply set the **$#A** to desired size. So for example,

```
@array = (10,12,45);
$#array = 1;
```

Will result in an array of size 2 or simply

```
@array = (10,12);
```

Control Structures (Loops and Conditionals)

There are various loop controls in perl. Here are some example.

A While Loop

```
$x = 1;
while ($x < 10){
    print "x is $x\n";
    $x++;
}
```

Until loop

```
$x = 1;
until ($x >= 10){
    print "x is $x\n";
    $x++;
}
```

Do-while loop

```
$x = 1;
do{
    print "x is $x\n";
    $x++;
} while ($x < 10);
```

for statement

```
for ($x=1; $x < 10; $x++){
    print "x is $x\n";
}
```

foreach statement

```
foreach $x (1..9) {
    print "x is $x\n";
}
```

There are variations to this code

```
@range1 = (1..5);
@range2 = (10,15..20);

foreach $i (@range1, @range2) {
   print $i;
}
```

Question: What would be the output of the above code?

Example: A perl program code that performs bubble sort on an array of strings is given below.

```
for ($i=0; $i<n; $i++)
{  for ($j=0; $j<n-$i-1; $j++)
    {  if ($arr[$j] gt $arr[$j+1])
        {
           $tmp = $arr[$j];
           $arr[$j]=$arr[$j+1];
           $arr[$j+1]=$tmp;
        }
    }
}
```

Exercise: Write a Perl program that performs binary search in the arr looking for the string variable $target

PERL I/O

Perl more or less work similar to other high level languages when it comes to file handling. Perl provides the standard file handlers such as STDIN, STDOUT, and STDERR.

Reading Data from STDIN

Interactive IO is input given to the perl program via STDIN and STDOUT. For example, we can read a line from the STDIN as follows.

$name = <STDIN>;

This variable $name contains the newline character that can be removed using,

chomp($name);

Reading Data from a File

Suppose we'd like to read a bunch of strings from a file into an array. Let us assume that we start with a default size of 10 and then doubles the size of the array when we need more space. We can accomplish the task as follows.

```
$size = 10;
open(INFILE, "file.txt");
$#arr = $size-1; # initialize the size of the array to 10
$i = 0;
foreach  $line (<INFILE>) {
   $arr[$i++] = $line;
   if ($i >= $size) {
      $#arr = 2*$#arr + 1; # double the size
      $size = $#arr + 1;
   }
}
```

Writing to a File

To open a file for writing is similar to open the file for reading except the file name is preceded by ">". For example

open(OUT, ">out.txt");

associates file "out.txt" with the file handle OUT so output can be written to this file. For example,

print OUT "hello there\n";

now prints the string "hello there" to the file out.txt

Warning: A file handle that is not successfully opened may not show any warnings and any read or write will result in no action. To make sure file was opened properly, we can use the "die" command as follows.

open (OUT, ">out.txt") || die "sorry out.txt could not be opened\n";

The function die gets executed only of open if false.

Example: The following Perl code reads from the passwd file and writes the passwords to an output file.

```
open(OUT, ">passwd.txt");
open(IN, "/etc/passwd");
while (<IN>){
   chomp;
   print OUT "$_\n";
}
```

We can search, sort, and pretty much do anything with an array as in other major programming languages. This is only a small sample of what perl programs can do. There is ton of stuff on the web for learning perl. A good reference for beginners is **http://www.perl.com/pub/a/2000/10/begperl1.html**

Regular Expressions in Perl

As we learnt in the previous lesson, regular expression is a pattern that defines a class of string that fits into the pattern. Perl has strong regex capabilities and that makes perl an ideal language to do tasks that require text parsing.
Suppose we need to read a file of html text and parse them into separate lines. Then we can think about how to parse individual words, tags and tokens within the html file. For example, consider my webpage, index.html (http://www.cs.cmu.edu/~guna) and list all the lines that contain the word "guna". We can accomplish this task by using a regular expression (regex). The perl code is:

```
#! /usr/local/bin/perl

open(INFILE, "index.html");
foreach $line (<INFILE>) {
  if ($line =~ /guna/ ){
    print $line; #read a line of text and chop the newline
  }
 }
close(INFILE);
```

and here is the output produced by the above code.

**
guna at cs dot cmu dot edu**
at pgh-lk website.
<td><img SRC="button_mail.gif" ></td>

So 3 lines matched (out of 312 lines in index.html) and each line has the substring "guna". The regular expression /guna/ indicates we are looking for any line that contains "guna" as a substring.

There are few things that are new in the above code. The regex is enclosed between / / and the **binding operator =~** in perl (negation is !~) is used to look for substring matching the expression enclosed within / /. We can do more with regular expressions.

For example, suppose we need to look for all the email tags within the html file. We know that mail tags typically is enclosed in a line that contains

"mailto:guna@cs.cmu.edu"

and what we need to do is to extract the string guna@cs.cmu.edu. We can then look for the regex

/mailto: (.*?)"/

This matches the expression of the form mailto:guna@cs.cmu.edu". The ? mark is there is a "lazy" match indicating as soon as the first " is matched, the regex ends. Without the ? sign, regex will continue to match and will find the last " in the input. The matching expression inside the parenthesis (.*?) is assigned the variable $1 and can be used to print out the actual email address as follows. Here is the sample code that opens "guna.htm" , read each line and look for a match to find the email addresses in each line.

```
open(IN, "guna.htm");
while (<IN>){
    if ($_ =~ /mailto:(.*?)"/){
        print $1."\n";
    }
}
```

We note that $_ is a system variable that stores the current input and **while (<IN>)** can be used to replace **foreach $line (<IN>)**

Exercise: Modify the code so that it extracts all http links from the webpage. Hint: "http:\/\/.*?"

Other Examples of Regex in Perl

1. /[abc]|[123]/ --- matches a string that contains [abc] or [123]
2. /\dguna/ --- matches a string that begins with a digit and followed by guna
3. /gu+n?a/ -- matches guna, gun, guuna, etc
4. $_ = “i like programming”; s/ +/_/; -- replaces consecutive spaces with _ character.
5. /g{3,8}/ -- matches 3 to 8 g’s
6. /g{2,}/ -- matches with strings with at least 2 g’s
7. /g{4}/ -- matches with strings with exactly 4 g’s

Exercises

1. Write a perl program that reads a list of n strings (from STDIN) into an array and print a random string from the list (use srand; rand(@array))
2. Write a perl program to read a list of n numeric’s from STDIN and find the max, min, range, median and mode. Input size of the list n interactively.
3. Write a perl program that starts with an array of size 1 and then doubles the size of the array when user needs more space. Program will read numbers from <STDIN> until the user types -999.